CS 3200

The Relational Model of Data

Mark Fontenot, PhD
Northeastern University

These notes are based upon and adapted via the generosity of Prof. Nate Derbinsky.

Reminders

- Make sure you can see the course on Gradescope, Campuswire, and Slack
- **<u>Mini HW00</u>**
  - Early EC Deadline: Sunday @ 11:59 pm
  - Regular Deadline: Tuesday @ 11:59 pm

Applications and Data

- Many applications are data intensive compared to *compute intensive*.
- Many apps need to…
  - store data for itself or another app to use in the future (*databases*)
  - remember the results of prior expensive operations (*caches*)
  - allow users to search for data efficiently (*indexes*)
  - send messages with data to another application to handle (*stream processing*)
  - periodically process a large accumulation of data (*batch processing*)

Stores of data?

What is a database?

- Structured collection of related data
  - usually related to something in the real world
  - usually created…
    - for a specific group of users
    - to help these users perform some kind of tasks
    - to hopefully complete those tasks with some ***performance***, ***redundancy***, ***concurrency***, and/or ***security considerations*** in mind

Notion

- Intended users?
- Intended tasks?
- Considerations of
  - Performance?
  - Concurrency?
  - Redundancy?
  - Security?

Database Management Systems (DBMS)

- Software that allow the *creation* and *maintenance* of databases
  - Support the encoding of some type of structure for the data
  - Persists the data
  - Support **adding** new data and **updating** existing data
  - Protects against failures and unauthorized access

Some Categories of DBMSs

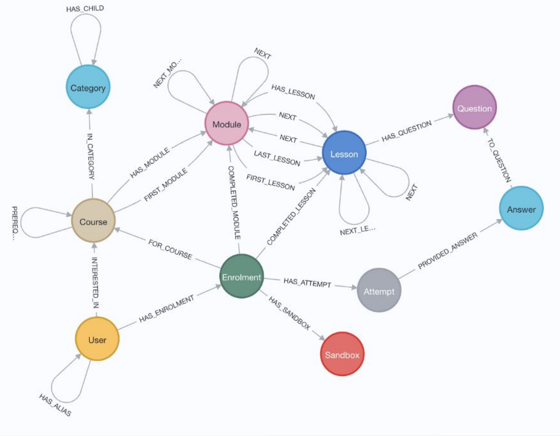
Document-Oriented Databases

- Organizes and queries data based on the concept of a “document” often in JSON
- usually considered semi-structured

```
{
    "count": 7,
    "items": ["socks", "pants", "shirts", "hats"],
    "manufacturer": {
        "name": "Molly's Seamstress Shop",
        "id": 39233,
        "location": {
            "address": "123 Pickleton Dr.",
            "city": "Tucson",
            "state": "AZ",
            "zip": 85705
        }
    },
    "total_price": "$393.23",
    "purchase_date": "2022-05-30",
    "country": "USA"
}
```

Graph Databases

- Organizes data by nodes, edges, labels
- Query about paths between nodes and node relationships



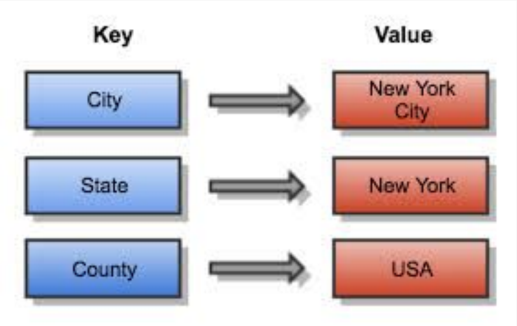
Some Categories of DBMSs

<u>Key-Value Databases</u>

- Everything is a key/value pair
- Based on associative array

FoundationDB

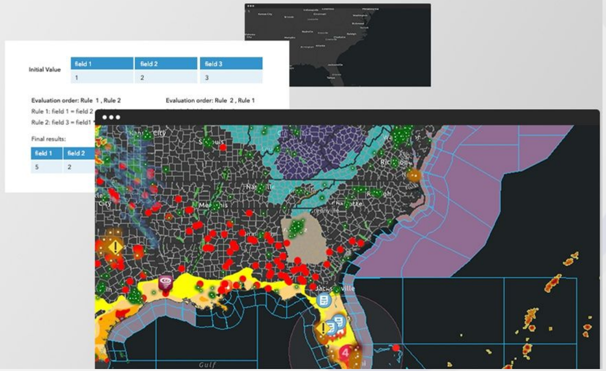
redis



<u>Spatial Databases</u>

- Stores data related to 2D/3D locations
- Query example: are 2 cars about to collide?

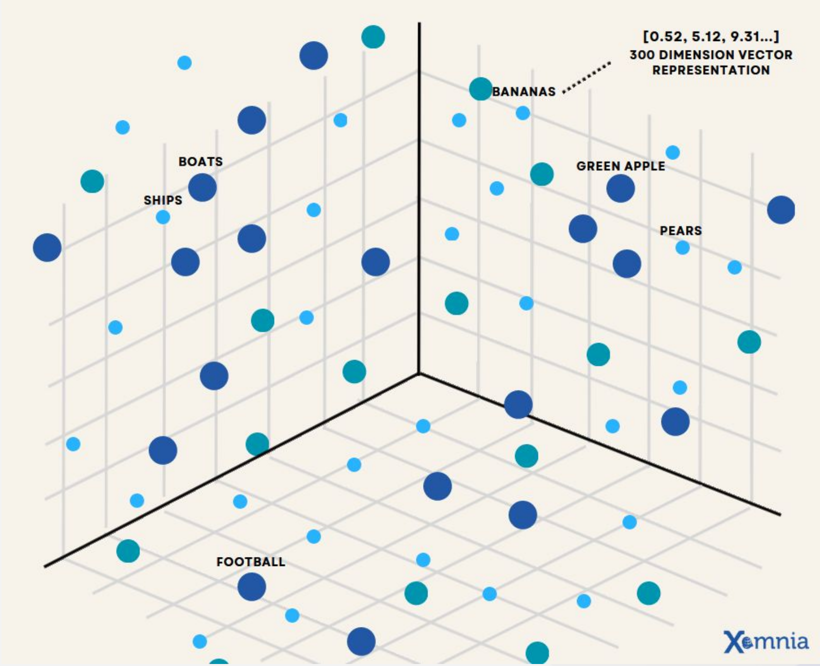
esri



Some Categories of DBMSs

Vector Databases

- unit of storage is a vector represent high-dimensional data
- highly performant similarity searches
- used extensively in LLMs



chroma

Milvus

LanceDB

The category for this course...

- **Relational Database** Management Systems
  - Based on storing data in tables and connections between those tables
  - Original concept developed in early 70s by EF Codd and colleagues

Relational Model of Data: Overview

The Relational Database: Relation/Table

Relation/Table Name

Attributes/Columns

Students

<u>ID</u>	Name	Phone	Dorm	Age	GPA
1123141	Mark	555-1234	1	19	3.21
2323411	Kim	555-9876	2	25	3.53
17642352	Sam	555-6758	1	19	3.25

Rows/ Tuples

Primary Key

<u>Relation</u> - the core construct in a relational database; collection of tuples with each tuple having values for a fixed number of attributes/fields.

<u>Relation Schema</u> - represents the attributes and their data types for a particular relation

<u>Relation Instance</u> - represents the state of the data in the relation at a particular point in time.

<u>Row/Tuple</u> - values for each relation's attribute for one element of a relation instance

Image borrowed from this Medium post.

The Relational Database: Constraints

Student

<u>ID</u>	Name	Phone	Dorm	Age	GPA
1123141	Mark	555-1234	1	19	3.21
2323411	Kim	555-9876	2	25	3.53
17642352	Sam	555-6758	1	19	3.25

Example Constraints:

cannot be null

Must be a valid dorm id in the dorm relation

Constraint - conditions that must hold on all valid relation instances

Types:
- Key Constraints
- Entity Integrity Constraints
- Referential Integrity constraints

Image borrowed from this Medium post.

The Relational Database: Relationships

Student

Name	ID	Phone	Dorm	Age	GPA
Mark	1123141	555-1234	1	19	3.21
Kim	2323411	555-9876	2	25	3.53
Sam	17642352	555-6758	1	19	3.25

Some values in one table are related (by design) to values in another table

Dorm

Dorm	Name
1	555 Huntington
2	Baker

Referential Integrity Constraint Examples

Class

ID	Class
1123141	COMP355
2323411	COMP355
17642352	MATH650
1123141	MATH650
2323411	BIOL110

The Relational Database: Queries

Student

Name	ID	Phone	Dorm	Age	GPA
Mark	1123141	555-1234	1	19	3.21
Kim	2323411	555-9876	2	25	3.53
Sam	17642352	555-6758	1	19	3.25

Dorm

Dorm	Name
1	555 Huntington
2	Baker

Class

ID	Class
1123141	COMP355
2323411	COMP355
17642352	MATH650
1123141	MATH650
2323411	BIOL110

Questions (Queries):

"Provide a list of student names and IDs with GPA between 3.0 and 3.5."

The Relational Database: Queries

Student

Name	<u>ID</u>	Phone	Dorm	Age	GPA
Mark	1123141	555-1234	1	19	3.21
Kim	2323411	555-9876	2	25	3.53
Sam	17642352	555-6758	1	19	3.25

Dorm

<u>Dorm</u>	Name
1	555 Huntington
2	Baker

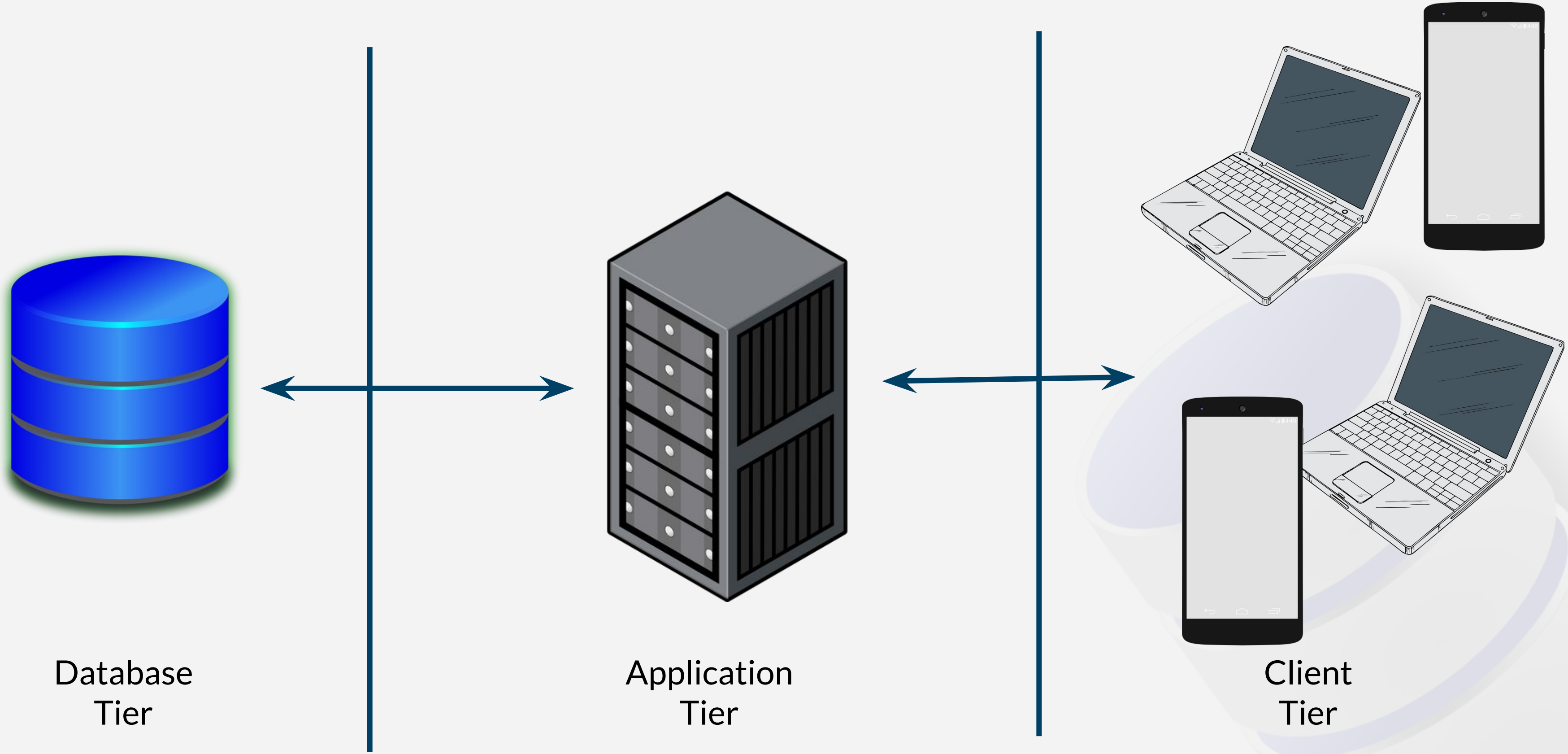
Class

<u>ID</u>	<u>Class</u>
1123141	COMP355
2323411	COMP355
17642352	MATH650
1123141	MATH650
2323411	BIOL110

Questions (Queries):

"Which students live in Baker Dorm?"

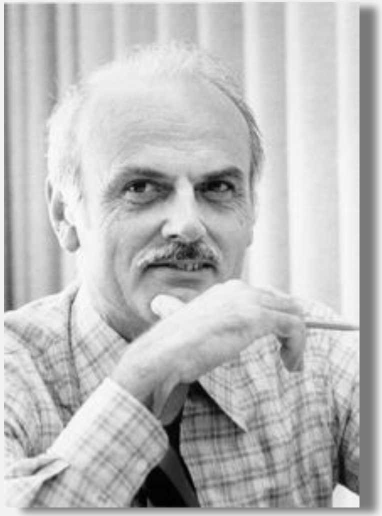
Databases in the Context of a Software System



The Relational Model of Data: Digging In

History 101

Codd, Edgar F. "A relational model of data for large shared data banks." Communications of the ACM 13.6 (1970): 377-387.



"Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation)…"

Information Retrieval

P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of data independence—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of data inconsistency which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed without logically impairing some application programs is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

Volume 13 / Number 6 / June, 1970 Communications of the ACM 377

History 101

- The **relational model** provides a *formal mathematical basis* for the structure of and interaction with a relational database
  - Based on set theory and first-order predicate logic
  - The formal basis allows for robust scientific development of the model.
- The (eternal) struggle of theory vs. practice…
  - Most modern RDBMSs don't strictly adhere to the purest mathematical formalisms in the relational model.

A Little Set Theory Review...

- What's a set?

  A **set** is a collection of unique objects. The *things* (*objects*) inside a set are called **elements**.

- What is the **cardinality** of a set? (denoted |S|)

  cardinality → the number of unique elements in the set.

?

What are some examples of sets?

A Little Set Theory Review...

A = {1, 3, 5} B = {3, 4, 5, 6}

- What is the **Union** of sets A and B (A ∪ B)?

AUB = {1, 3, 5, 4, 6}

- What is the **intersection** of sets A and B (A ∩ B)?

A∩B = {3, 5}

- What is the **set difference** of A and B (A - B)?

A-B = {1}

A Little Set Theory Review...

- What is the **cartesian product** of two sets?

The *cartesian product* of two sets A and B is the set of all ordered pairs (a, b) where a ∈ A and b ∈ B.

A = {123, 435}   B = {Chul, Kev, Sal}

- What is the cartesian product of A and B (aka A x B)?

```
AXB = { (123, Chul), (123, Kev), (123, Sal),
(435, Chul), (435, Kev), (435, Sal) }
```

What's a Relational Database?

- A **Relational Database** consists of
  - a collection of **relations**, and
  - a collection of **constraints.**
- A relational database is in a **valid/consistent state** if it satisfies all constraints (else, **invalid/inconsistent state**).

Relations (Some Review)

- A **relation** consists of
  - its **schema** → *a description of the structure of the relation* (<u>relation schema</u>)
  - its **state** → *the current data that is populated in the relation* (<u>relation instance</u>)
- The **schema** of a relation includes
  - the **name** of the relation
  - an list of *n* **attributes** each with an associated **domain** (what values that attribute can take on).
- Notation: *REL_NAME(Attrib1:Dom1, Attrib2:Dom2…)*

More formally...

- Let $A_1, A_2, \ldots, A_n$ be names of attributes of relation R with associated domains $D_1, D_2, \ldots, D_n$, then

  $$R(A_1{:}D_1, A_2{:}D_2, \ldots A_n{:}D_n)$$

  is a **relation schema** and n, the **degree** of R, represents the number of attributes of R.

- Then, an **instance** of Relation R is a subset of the cartesian product of the domains of the attributes of R.

Relations - Example

- Assume we have the following domains:
  - names → {'Jared', 'Sakshi'}
  - id_nums → {all 9 digit positive integers starting with 00}
  - majors → {'CS', 'DS', 'CY'}
- Defining the ***TA*** relation schema:

  TA(name: names, id: id_nums, major: majors)

?

Is the following a valid instance of TA?

```
{
    ('Jared', 001928374, 'CS')
    ('Sakshi', 001122334, 'DS')
}
```

Relations - Example

- Assume we have the following domains:
  - names → {'Jared', 'Sakshi'}
  - id_nums → {all 9 digit positive integers starting with 00}
  - majors → {'CS', 'DS', 'CY'}
- Defining the ***TA*** relation schema:

  TA(name: names, id: id_nums, major: majors)

? **Is the following a valid instance of TA?**

```
{
    ('Sakshi', 001928374, 'CS')
    ('Sakshi', 001122334, 'CY')
}
```

Relations - Example

- Assume we have the following domains:
  - names → {'Jared', 'Sakshi'}
  - id_nums → {all 9 digit positive integers starting with 00}
  - majors → {'CS', 'DS', 'CY'}
- Defining the ***TA*** relation schema:

  TA(name: names, id: id_nums, major: majors)

?

Is the following a valid instance of TA?

```
{
    ('Sakshi', 001928374, 'CS')
    ('Dylan',  001122334, 'DS')
}
```

Relations - Example

- Assume we have the following domains:
  - names → {‘Jared’, ‘Sakshi’}
  - id_nums → {all 9 digit positive integers starting with 00}
  - majors → {‘CS’, ‘DS’, ‘CY’}
- Defining the ***TA*** relation schema:

  TA(name: names, id: id_nums, major: majors)

? **Is the following a valid instance of TA?**

```
{
    ('Sakshi', 001928374, 'CS')
    ('Jared',  001122334)
}
```

Relation Instance

- A **relation instance** is a set of tuples (rows) from a relation at a particular point in time.
- Each **tuple (row)** is an ordered sequence of values, one for each attribute (possibly null)
  - usually enclosed in **<** and **>**

Student

Name	<u>ID</u>	Phone	Dorm	Age	GPA
Mark	1123141	555-1234	1	19	3.21
Kim	2323411	555-9876	2	25	3.53
Sam	17642352	555-6758	1	19	3.25

1 Tuple

Null Value

- **Null** is a special value that may exist in the domain of an attribute
- Could mean different things
  - value unknown
  - value unavailable right now
  - attribute doesn't apply to this tuple
- Does NOT mean:
  - zero (0)
  - the empty string ('')
- (NULL != NULL) Comparing two values of NULL does NOT return true

Value of an Attr in a Tuple

- Values should be **atomic**
  - Say NO to **composite attributes** (ex: address that includes city, state and zip)
  - Say NO to **multi-valued attributes** (ex: all email addresses for 1 person)

Student

Name	ID	Address	Phone	Dorm	Age	GPA
Mark	1123141	121 Anystreet Boston MA 02212	555-1234 555-1876	1	19	3.21
Kim	2323411	235 Huntington Boston MA 02215	555-9876	2	25	3.53

Super and Candidate Keys

- ***key*** - a subset of attributes of a relation used to uniquely identify each tuple

- A ***super key*** of a relation *R* is a subset of the attributes of *R* such that no two distinct tuples in any possible relation instance will have the same values for the subset of attributes.
  - may not be *minimal* - could contain attributes that aren't needed for unique determination
- A ***candidate key*** of relation *r* is a minimal super key.
  - A relation may have more than one candidate keys.

Keys - Superkey

Customers

customerNumber	customerName	contactLastName	contactFirstName	salesRepEmployeeNumber
103	Atelier graphique	Schmitt	Carine	1370
112	Signal Gift Stores	King	Jean	1166
114	Australian Collectors, Co.	Ferguson	Peter	1611
119	La Rochelle Gifts	Labrune	Janine	1370
121	Baane Mini Imports	Bergulfsen	Jonas	1504
124	Mini Gifts Distributors Ltd.	Nelson	Susan	1165
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	<null>
128	Blauer See Auto, Co.	Keitel	Roland	1504
129	Mini Wheels Co.	Murphy	Julie	1165
131	Land of Toys Inc.	Lee	Kwai	1323
141	Euro+ Shopping Channel	Freyre	Diego	1370

Some Possible Superkeys:

(customerNumber, customerName)

(customerNumber, salesRepEmployeeNumber)

(customerNumber)

Not Minimal

Minimal

Keys - Candidate Keys

Customers

customerNumber	customerName	contactLastName	contactFirstName	salesRepEmployeeNumber
103	Atelier graphique	Schmitt	Carine	1370
112	Signal Gift Stores	King	Jean	1166
114	Australian Collectors, Co.	Ferguson	Peter	1611
119	La Rochelle Gifts	Labrune	Janine	1370
121	Baane Mini Imports	Bergulfsen	Jonas	1504
124	Mini Gifts Distributors Ltd.	Nelson	Susan	1165
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	<null>
128	Blauer See Auto, Co.	Keitel	Roland	1504
129	Mini Wheels Co.	Murphy	Julie	1165
131	Land of Toys Inc.	Lee	Kwai	1323

Some Possible Superkeys:

(customerNumber, customerName)

(customerNumber, salesRepEmployeeNumber)

(customerNumber)

Not Candidate Keys

Candidate Key

The Primary Key

- The ***primary key (PK)*** of relation *R* is chosen from the set of candidate keys
  - If a relation has only 1 candidate key, it becomes the PK.
  - If a relation has > 1 candidate key, the database designer chooses one based on business requirements
  - Every relation must have a PK
  - *Entity Integrity Constraint* → PK values must be unique and may NOT be null
  - (Usually) the PK is underlined in a relation schema or table

Keys - Primary Key

Customers

customerNumber	customerName	contactLastName	contactFirstName	salesRepEmployeeNumber
103	Atelier graphique	Schmitt	Carine	1370
112	Signal Gift Stores	King	Jean	1166
114	Australian Collectors, Co.	Ferguson	Peter	1611
119	La Rochelle Gifts	Labrune	Janine	1370
121	Baane Mini Imports	Bergulfsen	Jonas	1504
124	Mini Gifts Distributors Ltd.	Nelson	Susan	1165
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	<null>
128	Blauer See Auto, Co.	Keitel	Roland	1504
129	Mini Wheels Co.	Murphy	Julie	1165
131	Land of Toys Inc.	Lee	Kwai	1323
141	Euro+ Shopping Channel	Freyre	Diego	1370

Some Possible Superkeys:

~~(customerNumber, customerName)~~

~~(customerNumber, salesRepEmployeeNumber)~~

(customerNumber)

Chosen as Primary Key

If there are 2+ Candidate Keys, DB Designer will choose one as the Primary key

Foreign Keys

- **Foreign Key** (FK) - An attribute a_i in one relation R_C (the child relation) refers to/references the PK a_j in another relation R_P (parent relation) such that all values of a_i must either be NULL or contain a value from a_j.

- <u>Self-Referential Relation</u> → R_C and R_P are the same relation
- Foreign Key == <u>Referential Integrity Constraint</u>
- Foreign Keys are the operationalization of relationships in a relational database

Foreign Key Example

Note: in this example, Grades.sID cannot contain null values because it is part of the PK of Grades Relation.

Primary Keys

Foreign Key

Students

<u>sID</u>	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Grades

<u>cID</u>	<u>sID</u>	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

Parent Relation

(the one being referenced)

Child Relation

(the one referencing another relation)

FKs - but why???

Customers

	custNum	custName	custLast	custFirst	empNum	empLast	empFirst	empEmail
1	114	Australian Collectors, Co.	Ferguson	Peter	1611	Fixter	Andy	afixter@classicmodelcars.com
2	166	Handji Gifts& Co	Victorino	Wendy	1612	Marsh	Peter	pmarsh@classicmodelcars.com
3	276	Anna's Decorations, Ltd	O'Hara	Anna	1611	Fixter	Andy	afixter@classicmodelcars.com
4	282	Souveniers And Things Co.	Huxley	Adrian	1611	Fixter	Andy	afixter@classicmodelcars.com
5	323	Down Under Souveniers, Inc	Graham	Mike	1612	Marsh	Peter	pmarsh@classicmodelcars.com
6	333	Australian Gift Network, Co	Calaghan	Ben	1611	Fixter	Andy	afixter@classicmodelcars.com
7	357	GiftsForHim.com	MacKinlay	Wales	1612	Marsh	Peter	pmarsh@classicmodelcars.com
8	412	Extreme Desk Decorations, Ltd	McRoy	Sarah	1612	Marsh	Peter	pmarsh@classicmodelcars.com
9	471	Australian Collectables, Ltd	Clenahan	Sean	1611	Fixter	Andy	afixter@classicmodelcars.com
10	496	Kelly's Gift Shop	Snowden	Tony	1612	Marsh	Peter	pmarsh@classicmodelcars.com

Notice: empLast, empFirst, and empEmail attributes contain duplicated/repeated data. Opens up the possibility of *insert* or *update* anomalies.

So, put them in a separate table along with empNum and refer back to the new table (becomes the parent table)

Relational Algebra

Relational Algebra

- *Relational Algebra (RA)* : a procedural query language for relations that allow us to retrieve information from a relational database
  - A *query* is an operation or set of operations applied to one or more relation instances
  - RA is **closed**, meaning the result/output of each query is another relation
  - In RA, order of operations matters
- Not a full-fledged (turing complete) programming language

Quick Aside: *Predicate Functions*

- *Predicate functions* - functions that return **true** or **false**.
- We will use predicate functions (or just “predicates”) to determine if tuples in a relation instance should be returned by a query or not.
  - in the form *attr <op> attr* OR *attr <op> <constant>*
  - *<op>* can be any standard relational operator (=, !=, <, >, <=, etc)
  - predicates can be composed with **^** (and), **v** (or), **¬** (not).

Employees

empID	firstName
333	Bob
143	Sam

Examples:

- `empID = 143`
- `empID > 400`
- `firstName = ‘Bob’ ^ empID = 333`

The First 8 Basic Operations of RA

Highest Precedence

1.	Select	σ	(Tuple Filtering)
2.	Project	π	(Attribute Filtering)
3.	Rename	ρ	
4.	Cartesian Product	$\times$	
5.	Join	$\bowtie$	
6.	Intersection	$\cap$	
7.	Set-difference	$-$	
8.	Union	$\cup$	

Lowest Precedence

You can always use ( ) to change the order of operations.
Often times, ( ) make things more clear.

Relational Algebra: Select Operator

- **Select** - return a relation containing tuples from relation *R* that satisfy predicate *pred*.

Notation: $\sigma_{pred}(R)$

- Think of it as a *horizontal subset* (subset of tuples/rows) of a relation instance

Relational Algebra: Select Operator

$$\sigma_{(Enrollment>500)}(Enrollments)$$

Enrollments

Dept	Class	Enrollment
CS	3200	40
CS	2500	643
CS	1800	680
DS	2000	412

Result:

Enrollments

Dept	Class	Enrollment
CS	2500	643
CS	1800	680

Relational Algebra: Select Operator

$$\sigma_{(Dept=Taught\_by \wedge Class>3000)}(Enrollments)$$

Enrollments

Dept	Class	Taught_by
CS	3200	CS
CS	2500	CS
CS	1800	Math
DS	2000	Math

Result:

Enrollments

Dept	Class	Taught_by
CS	3200	CS

Relational Algebra: Project Operator

Project - returns a relation with a subset of attributes (A_1… A_k) from R.

Notation: $$\pi_{(A_1,\ldots,A_k)}(R)$$

Duplicate tuples will be removed from the resulting relation (because relations are sets).

Relational Algebra: Project Operator

Enrollments

Dept	Class	Enrollment
CS	3200	40
CS	2500	643
CS	1800	680
DS	2000	412

$$\pi_{(Dept,Class)}(Enrollments)$$

Result:

Enrollments

Dept	Class
CS	3200
CS	2500
CS	1800
DS	2000

$$\pi_{(Dept)}(Enrollments)$$

Result:

Enrollments

Dept
CS
DS

Relational Algebra: Cartesian Product

$$R \times S$$

Same operation from set theory.

Note: we can always use *Relation.Attribute* notation to resolve naming collisions.

Relations can't have two attributes with the same name

R

Attr_1	Attr_2
123	abc
456	def

S

Attr_1	Attr_3
123	CS
789	DS
111	Cyber

$R \times S$

R.Attr_1	R.Attr_2	S.Attr_1	S.Attr_3
123	abc	123	CS
123	abc	789	DS
123	abc	111	Cyber
456	def	123	CS
456	def	789	DS
456	def	111	Cyber

Union, Intersection & Difference

- Essentially the same as what we know from set theory.

$$R_1 \cup R_2 \qquad\qquad R_1 \cap R_2 \qquad\qquad R_1 - R_2$$

- One small difference: relations must be **schema compatible**
  - same number of attributes
  - attributes' domains must be compatible

More Complex RA Expressions

- Simple RA expression can be composed into more complex expressions
  - Remember: output of each RA operation is another relation

$$\sigma_{(a=b)}(R \times S)$$

$$\sigma_{(a=b)}(\pi_{(a,b,c)}(R))$$

Relational Algebra: Temporary Relation Names

For more complex RA queries, you can have:

- one long query expression
- an ordered list of smaller expressions, the result of each is given a temporary name with the ← operator

Example:

- TEMP_NAME ← $\pi_{(Dept)}(Enrollments)$
  TEMP_NAME can then be used as a relation in subsequent steps of the same query.
- Be careful about attribute naming collisions

Relational Algebra: Rename Operator

Rename Operator (rho) – Allows us to "rename" a relation, the attributes of a relation, or both.

- If only name is provided, the relation is being renamed (all attributes retain their original name.) $\rho_{employees\_data}(Employees)$

- List of attributes in parentheses means renaming attributes, but not relation (assume attributes originally (employeeID, lastName, firstName)

$$\rho_{(empID, empLast, empFirst)}(Employees)$$

- Rename both.

$$\rho_{employees\_data(empID, empLast, empFirst)}(Employees)$$

Developing Relational Algebra Expressions

Writing RA Queries

- Sometimes we need to evaluate an RA query against a database instance
  - Result is usually another relation instance/set of tuples/table
- Other times we need to convert the narrative form of a query into a RA query
  - Example: “Provide a list of all info from the Employee relation where the empID is less than 400.”
    - Answer: $\sigma_{empID<400}(Employees)$

Writing RA Queries

Students

Stu_ID	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Courses

Cou_ID	Dept	Number	Prof
332	CS	3345	Lawrimore
221	CS	2341	Fontenot
535	MATH	2339	Norris

Grades

C_ID	S_ID	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

Next few examples use this database schema.

Writing RA Queries

Students

Stu_ID	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Courses

Cou_ID	Dept	Number	Prof
332	CS	3345	Lawrimore
221	CS	2341	Fontenot
535	MATH	2339	Norris

Grades

C_ID	S_ID	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

Write a RA query that returns the names of all students.

$$\pi_{Name}(Students)$$

Writing RA Queries

Students

<u>Stu_ID</u>	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Courses

<u>Cou_ID</u>	Dept	Number	Prof
332	CS	3345	Lawrimore
221	CS	2341	Fontenot
535	MATH	2339	Norris

Grades

<u>C_ID</u>	<u>S_ID</u>	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

Provide a list of all course numbers taught by the CS Department.

$$\pi_{(Number)}(\sigma_{(Dept='CS')}(Courses))$$

Writing RA Queries

Students

<u>Stu_ID</u>	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Courses

<u>Cou_ID</u>	Dept	Number	Prof
332	CS	3345	Lawrimore
221	CS	2341	Fontenot
535	MATH	2339	Norris

Grades

<u>C_ID</u>	<u>S_ID</u>	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

List all 2nd year student names.

$$\pi_{(Name)}(\sigma_{(Year=2)}(Students))$$

Writing RA Queries

Grades

C_ID	S_ID	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

Students

Stu_ID	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Courses

Cou_ID	Dept	Number	Prof
332	CS	3345	Lawrimore
221	CS	2341	Fontenot
535	MATH	2339	Norris

What course or courses (dept and number) are taught by Professor Norris?

$$\pi_{(Dept,Number)}(\sigma_{(Prof='Norris')}(Courses))$$

Writing RA Queries

Students

Stu_ID	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Courses

Cou_ID	Dept	Number	Prof
332	CS	3345	Lawrimore
221	CS	2341	Fontenot
535	MATH	2339	Norris

Grades

C_ID	S_ID	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

List the letter grades that Sam has earned (don't need to include course info).

$$\pi_{(Grade)}(\sigma_{(Stu\_ID=S\_ID \wedge Name='Sam')}(Students \times Grades))$$

or $$\pi_{(Grade)}(\sigma_{(Name='Sam')}(\sigma_{(Stu\_ID=S\_ID)}(Students \times Grades)))$$

Joining Data from Two Relations

Relational Algebra: The Join Operator

- **Join** - allows us to combine data from two relations.
  - Subset of the cartesian product of the two argument relations based on explicit or implicit join predicate.

- 2 Versions:
  - Natural Join - Join relations on attributes with the same name

$$R \bowtie S$$

  - Theta Join (or *condition join* or simply *Join*) - Join relations with explicitly supplied join predicate

$$R \bowtie_\theta S$$

Natural Join

- Given: *A(R, S, T, U, V)* and *B(S, V, W, X)*
- Query: $A \bowtie B$
- Notice: *A* and *B* both have attributes named *S* & *V*.
- Result:
  - Schema of resulting relation is $attr(A) \cup attr(B)$ where *attr(R)* returns a set containing the attributes of R
    - so, attributes used in the implicit join condition are not duplicated in the result
  - contains any tuple from *A x B* where values for attributes *S* and *V* are equal
  - If A & B have no common attribute names, result is *A X B*.

Example

Students

sID	Name	Year
134	Chul	1
234	Kev	3
332	Sam	2
336	Ashwin	2

Grades

cID	sID	Grade
332	234	B
332	332	A
535	336	C
221	134	A
535	332	C
221	332	A
221	336	B

Query: $Students \bowtie Grades$

Result:

cID	sID	Grade	Name	Year
332	234	B	Kev	3
332	332	A	Sam	2
535	336	C	Ashwin	2
221	134	A	Chul	1
535	332	C	Sam	2
221	332	A	Sam	2
221	336	B	Ashwin	2

theta-Join (or Join... or Condition(al) Join)

- Operator has an explicit *join predicate (condition)* (doesn't rely upon attribute names)
- Result is a subset of the cartesian product *where provided predicate holds true*
- Assume: *S(sID, name)* and *G(stu-ID, course, semester, grade)*
- Query: $S \bowtie_{(S.sID=G.stu-ID)} G$
- Result:
  - Relation with schema (sID, name, stu-ID, course, semester, grade)
  - all tuples where S.sID = G.stu-ID
- Note: join condition can use other operators besides =.

theta-Join (or Join... or Condition(al) Join)

S

<u>A</u>	B
1	Cat
2	Dog
3	Bird

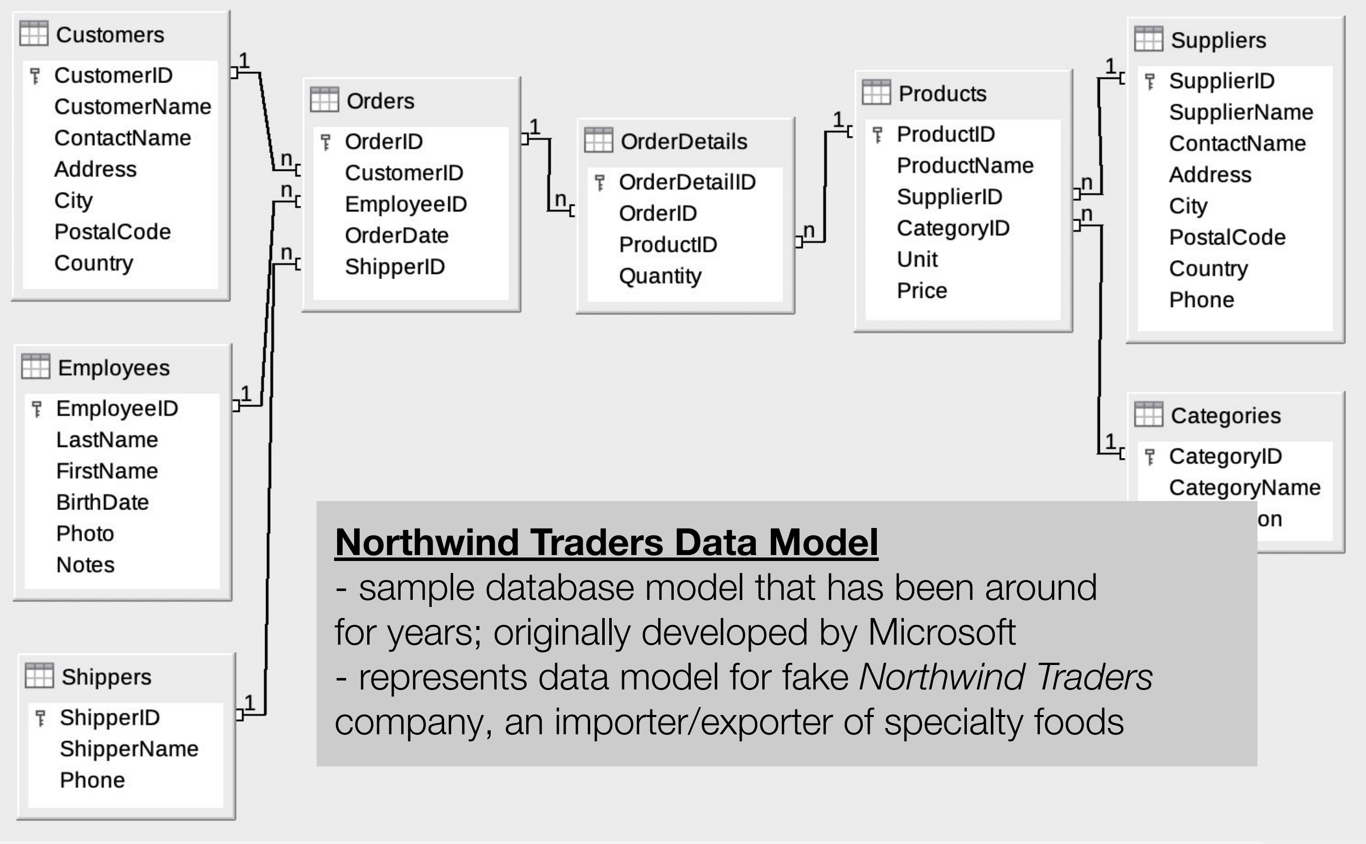
T

<u>C</u>	D
1	Meow
3	Chirp

$$S \bowtie_{(S.A=T.C)} T$$

A	B	C	D
1	Cat	1	Meow
3	Bird	3	Chirp

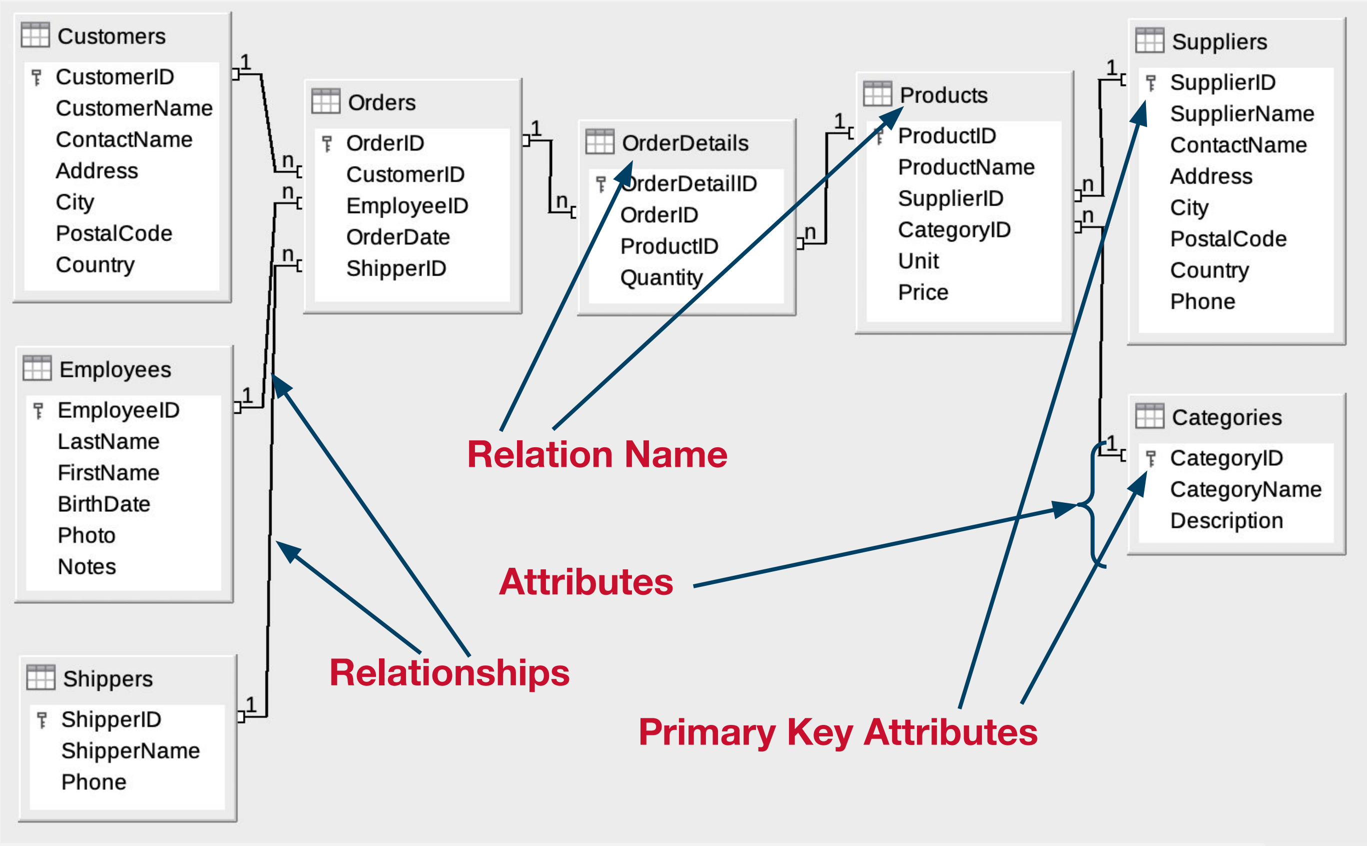
New: Entity Relationship Diagram



<u>Northwind Traders Data Model</u>

- sample database model that has been around for years; originally developed by Microsoft
- represents data model for fake *Northwind Traders* company, an importer/exporter of specialty foods

New: ER Diagram



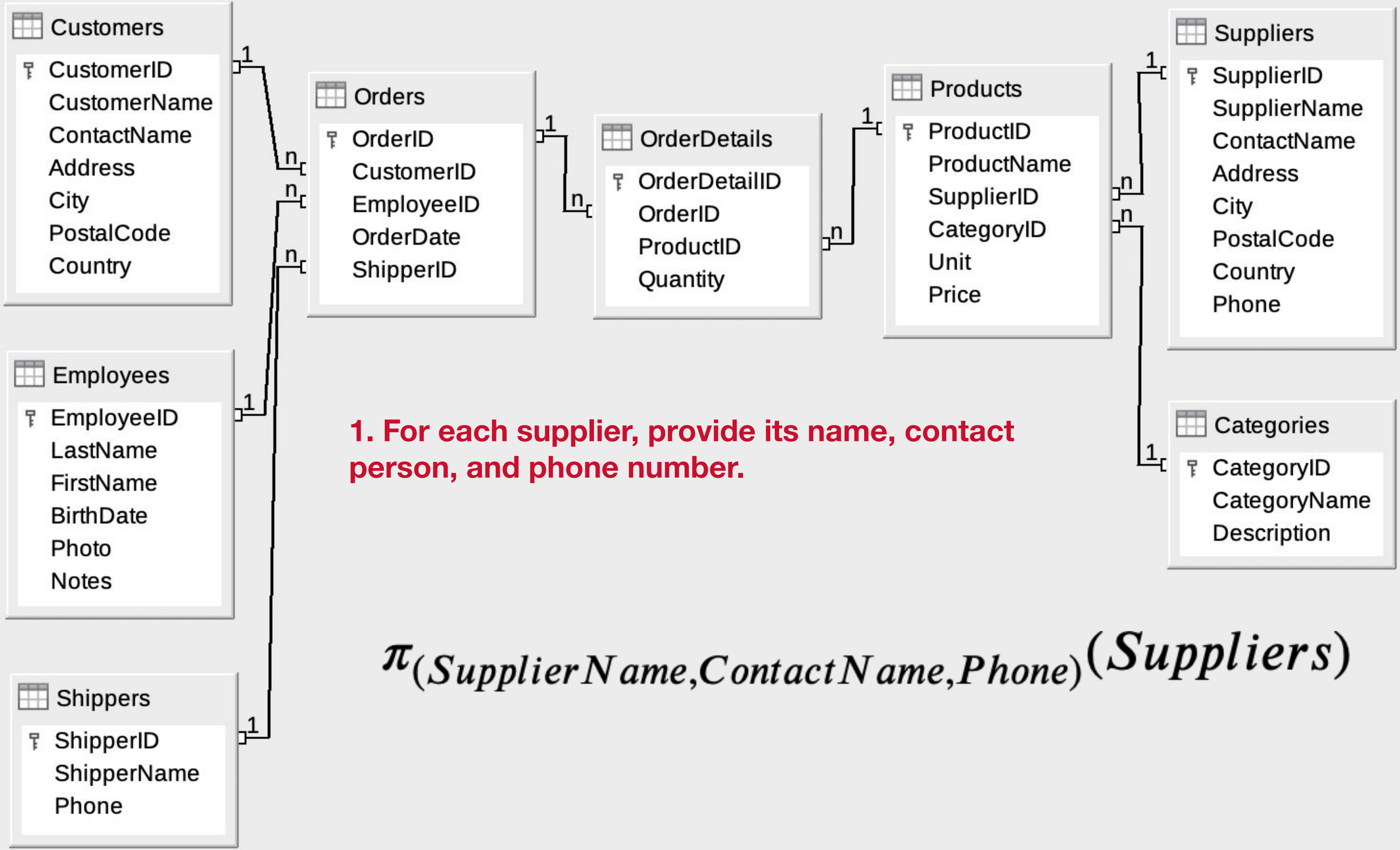
Relations in Northwind

- **Suppliers**: Suppliers and vendors of Northwind
- **Customers**: Customers who buy products from Northwind
- **Employees**: Employee details of Northwind traders
- **Products**: Product information
- **Shippers**: The details of the shippers who ship the products from the traders to the end-customers
- **Orders** and **Order_Details**: Sales Order transactions taking place between the customers & the company
- **Categories**: The categories a product can fall into

Practice Time!

For each of the following, compose a relational algebra query that satisfies the query prompt.

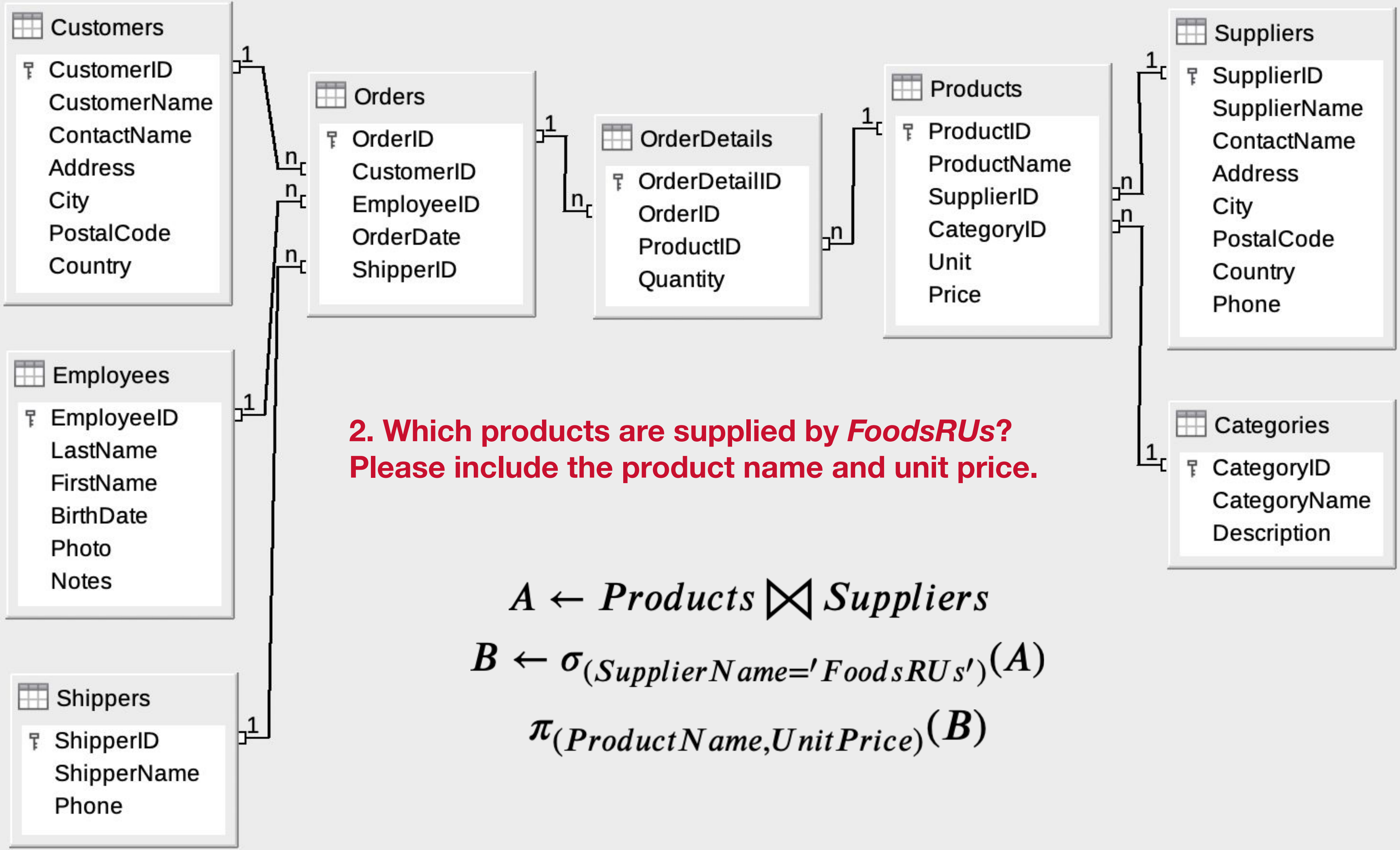
Query Time!



1. For each supplier, provide its name, contact person, and phone number.

$$\pi_{(SupplierName, ContactName, Phone)}(Suppliers)$$

Query Time!



2. Which products are supplied by *FoodsRUs*? Please include the product name and unit price.

$$A \leftarrow Products \bowtie Suppliers$$

$$B \leftarrow \sigma_{(SupplierName='FoodsRUs')}(A)$$

$$\pi_{(ProductName, UnitPrice)}(B)$$

Customers
- CustomerID
- CustomerName
- ContactName
- Address
- City
- PostalCode
- Country

Orders
- OrderID
- CustomerID
- EmployeeID
- OrderDate
- ShipperID

OrderDetails
- OrderDetailID
- OrderID
- ProductID
- Quantity

Products
- ProductID
- ProductName
- SupplierID
- CategoryID
- Unit
- Price

Suppliers
- SupplierID
- SupplierName
- ContactName
- Address
- City
- PostalCode
- Country
- Phone

Employees
- EmployeeID
- LastName
- FirstName
- BirthDate
- Photo
- Notes

Categories
- CategoryID
- CategoryName
- Description

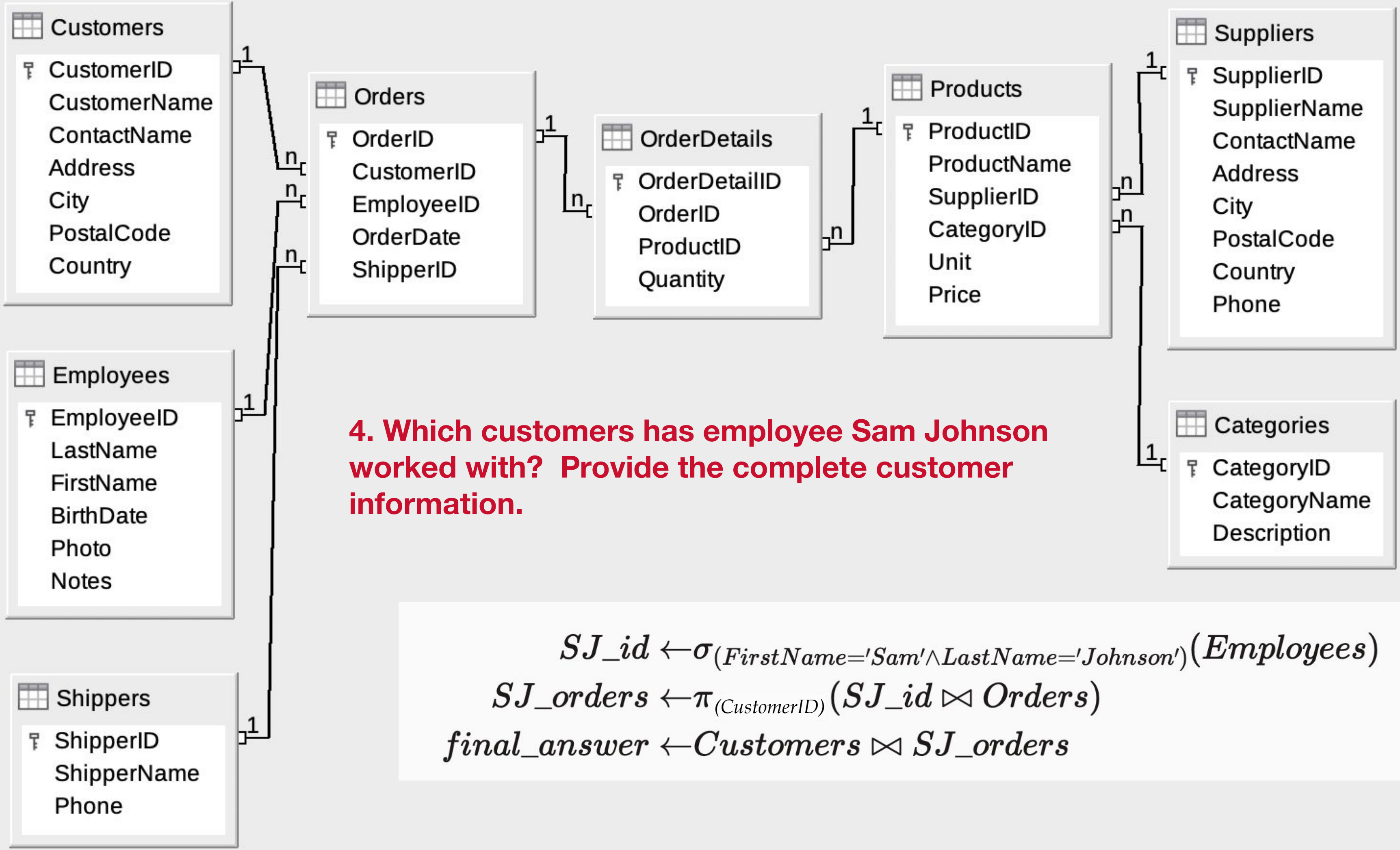
Shippers
- ShipperID
- ShipperName
- Phone

3. Provide a list of all product names ordered by *World Market* (a customer's name).

$$CustProd \leftarrow Customers \bowtie Orders \bowtie OrderDetails \bowtie Products$$

$$\pi_{(ProductName)}(\sigma_{(CustomerName='WorldMarket')}(CustProd))$$

Query Time!



4. Which customers has employee Sam Johnson worked with? Provide the complete customer information.

$$SJ\_id \leftarrow \sigma_{(FirstName='Sam' \wedge LastName='Johnson')}(Employees)$$

$$SJ\_orders \leftarrow \pi_{(CustomerID)}(SJ\_id \bowtie Orders)$$

$$final\_answer \leftarrow Customers \bowtie SJ\_orders$$

Further Reading

- Harrington Ch5 (OReilly)
- Foundations of Computer Science - Ch 8 - The Relational Model